CE888: Data Science and Decision Making
Storytelling through Unsupervised Learning

Ana Matran-Fernandez
amatra@essex.ac.uk
University of Essex

About

- We will be discussing ways of understanding the data
  - Without assuming there is something to predict
- For example, you might want to split your customers into different groups
  - But you have no idea which groups are out there
- You just have a description (i.e., features) of each sample
  - For example a collection of $< sales, location.... >$ tuples

How often is it done?

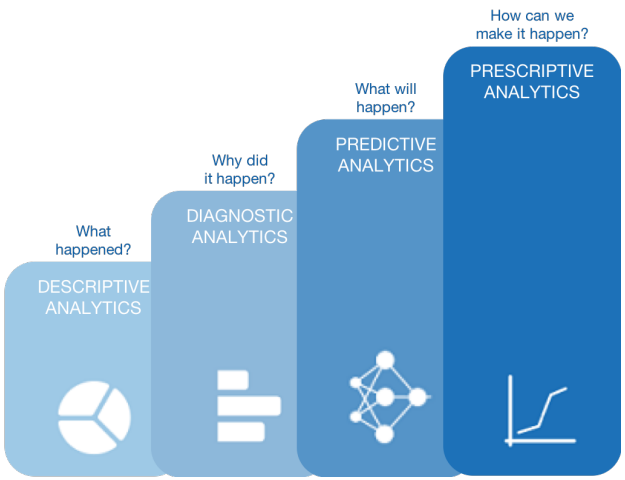
- Most of the data available is unlabeled
  - Labels are costly, tedious to obtain, and typically a lengthy process
- Our brains automatically group things all the time
  - Short / tall
  - Fast / slow
  - Clothing styles

Exploring data

"If intelligence was a cake, unsupervised learning would be the cake, supervised learning would be the icing on the cake, and reinforcement learning would be the cherry on the cake. We know how to make the icing and the cherry, but we don't know how to make the cake."

– Yann LeCun (Director of AI Research, Meta)

Data Science Operations



- Descriptive
  - "I'll create high level views of your data"
  - Also called analytics
- Predictive
  - "I'll try to predict a possible version of your future".
  - Also called...?
- Prescriptive
  - "What should I do to achieve certain results?"
  - Reinforcement Learning and Causality

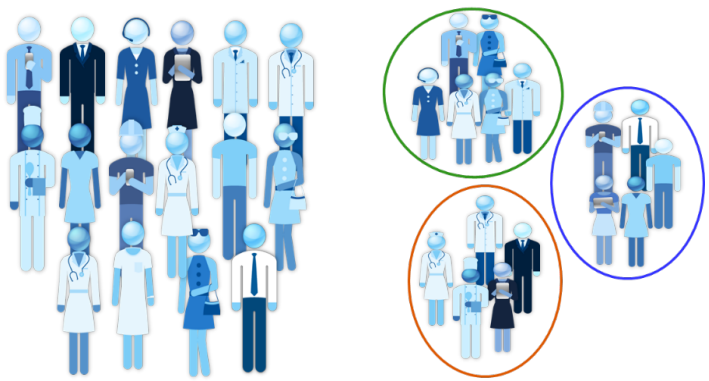
[1] Chris Wiggins. "Lectures delivered Aug 8-9, 2016 at MLSS.cc (Arequipa, Peru)". https://www.slideshare.net/chrishwiggins/machine-learning-summer-school-2016/75

IMPORTANT REMINDERS

- This week we have demos in the lab
- You **must upload** all the code you used to FASER **and demonstrate** it in the lab to me or a GLA
- Otherwise, you'll get a mark of 0
- Demos will take 5-10 minutes/student
  - Be prepared for the GLA when they approach you: **have your code ready**
  - The rest of the time work on this week's Quiz
- Demos will be in Lab 1 only

Clustering

- We would like to assign our data to different groups
- Are there "natural classes" in the data?



Clustering applications

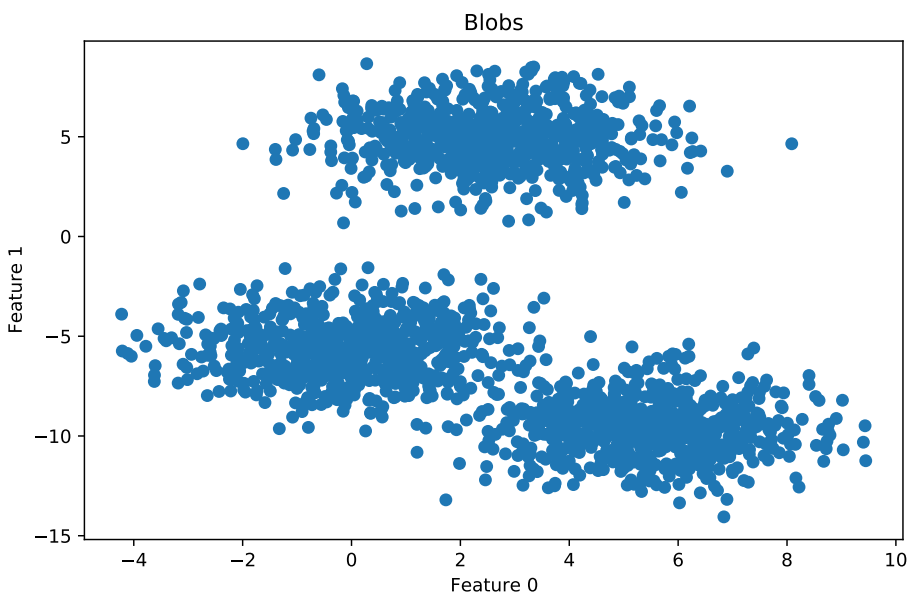
- Customer segmentation
- Data analysis
- Dimensionality reduction
- Anomaly/outlier detection
- Semi-supervised learning
- Image segmentation

The clustering problem

- To partition n observations (d-dimensional feature vectors) into groups (clusters)
- Different algorithms let us specify different parts of the problem
  - E.g., in K-Means we can say how many groups

Blobs dataset

```
from sklearn.datasets import make_blobs

X_blobs, y_blobs = make_blobs(n_samples=2000, n_features=2, centers=None, cluster_std=1.5,
                              center_box=(-10.0, 10.0), shuffle=True, random_state=10,
                              return_centers=False)
# Let's plot the blobs
plt.figure(figsize=(8,5))
plt.title("Blobs", fontsize=12)
plt.scatter(X_blobs[:, 0], X_blobs[:, 1])
plt.xlabel('Feature 0')
plt.ylabel('Feature 1')
plt.show()
```

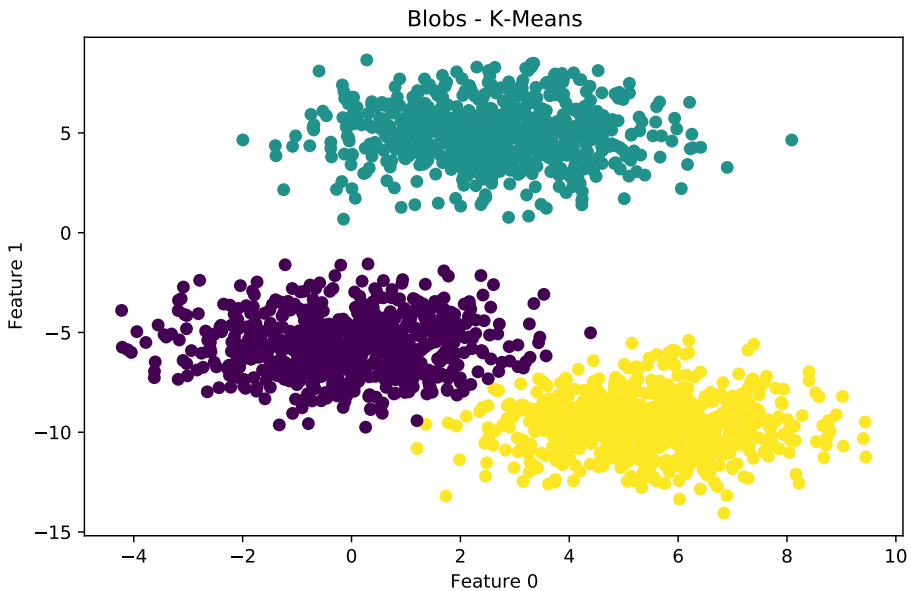


K-Means

- Possibly the most popular algorithm for clustering
- Aims to partition n observations (d-dimensional vectors) into k **clusters**
- 'Label' is the index of the cluster that the instance gets assigned to
  - Don't confuse it with 'label' in supervised learning!
  - Cluster assignments change with multiple runs of the algorithm
- Algorithm:
  1. Initialise with $n\_clusters$ random "centroids"
  2. Assign each point to the centroid to which it's closest
  3. Compare assignment with previous:
     - 3.1 If current assignment == previous assignment, end
     - 3.2 Else, recompute centroids and return to step 2

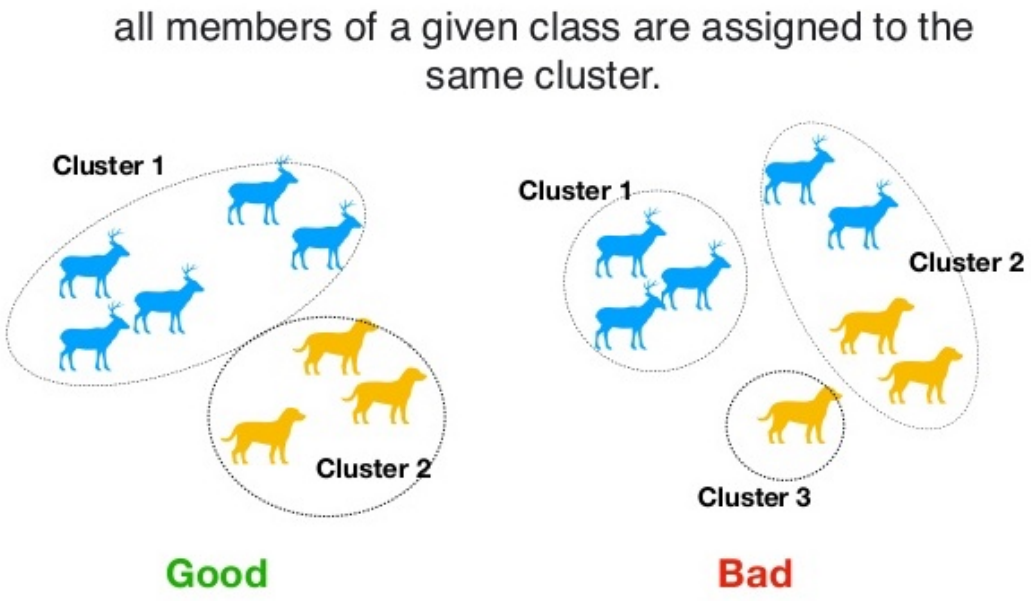
K-Means on Blobs

```
kmeans_blobs = KMeans(n_clusters=3)
y_pred_blobs = kmeans_blobs.fit_predict(X_blobs)

plt.figure(figsize=(8,5))
plt.title("Blobs - K-Means", fontsize=12)
plt.scatter(X_blobs[:, 0], X_blobs[:, 1], c=y_pred_blobs)
plt.xlabel('Feature 0')
plt.ylabel('Feature 1')

# Completeness score
print("Completeness score=%.3f" % completeness_score(y_blobs, y_pred_blobs))  # 0.967
# Silhouette score
print("Silhouette score=%.3f" % silhouette_score(X_blobs, y_pred_blobs))  # 0.664
```



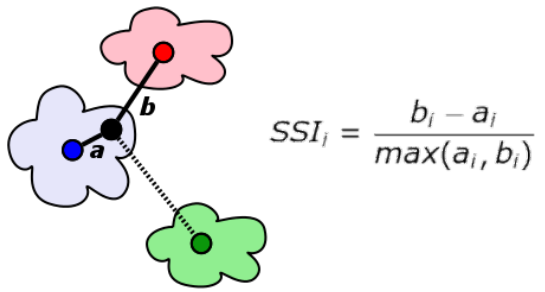
Clustering metrics: Completeness

- A clustering result satisfies completeness if all the data points that are members of a given class are elements of the same cluster.
  - It needs to know the ground truth label!
- Similar to accuracy, but it's independent of the values of the labels.
- Values $\in [0, 1]$.



Clustering metrics: Silhouette Coefficient

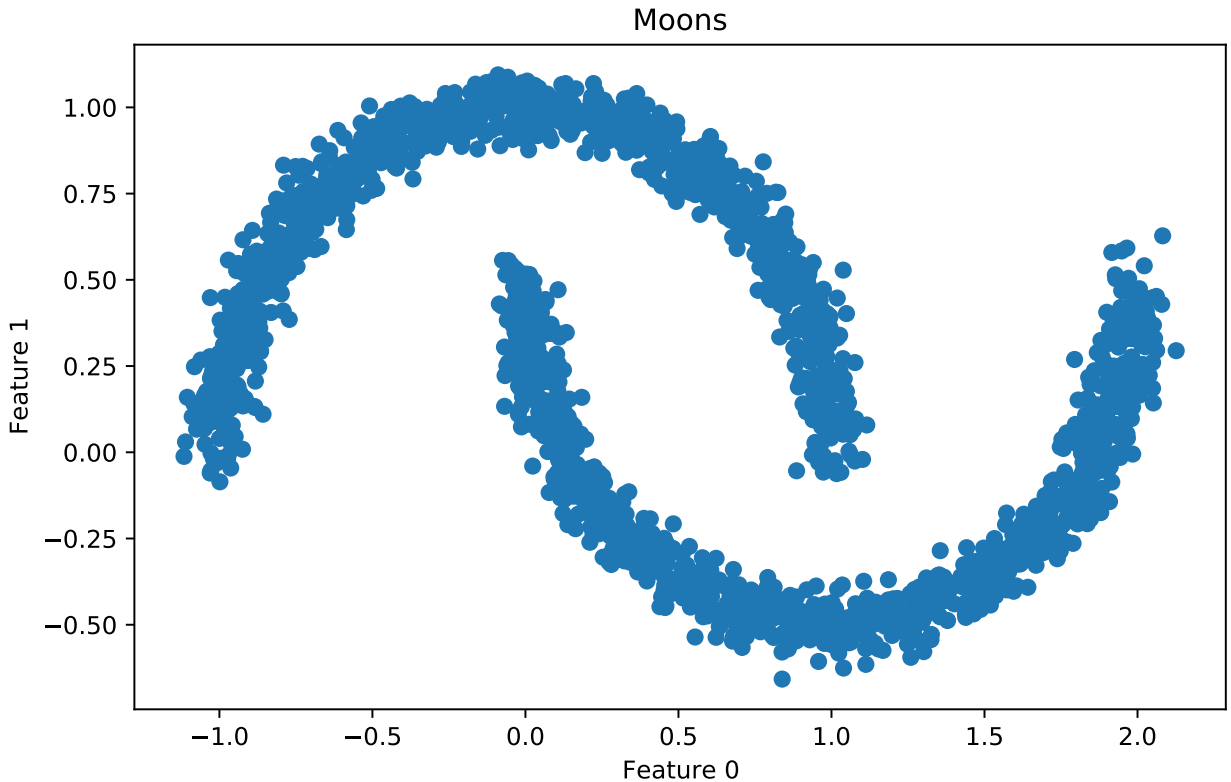
- For each data point i,
  - a_i is the mean distance between i and all the other points in the same cluster
  - b_i is the mean distance between i and all the points in the nearest/neighboring cluster of i



- Values $\in [-1, 1]$.
  - $s_i = 1$ indicates that i is far from the neighboring cluster
  - $s_i = 0$ indicates that i is on or very close to the decision boundary between two neighboring clusters
  - $s_i = -1$ indicates that i might have been assigned to the wrong cluster
- The average over all points is the Silhouette coefficient
  - $SSI = \frac{1}{n} \sum_i s_i$

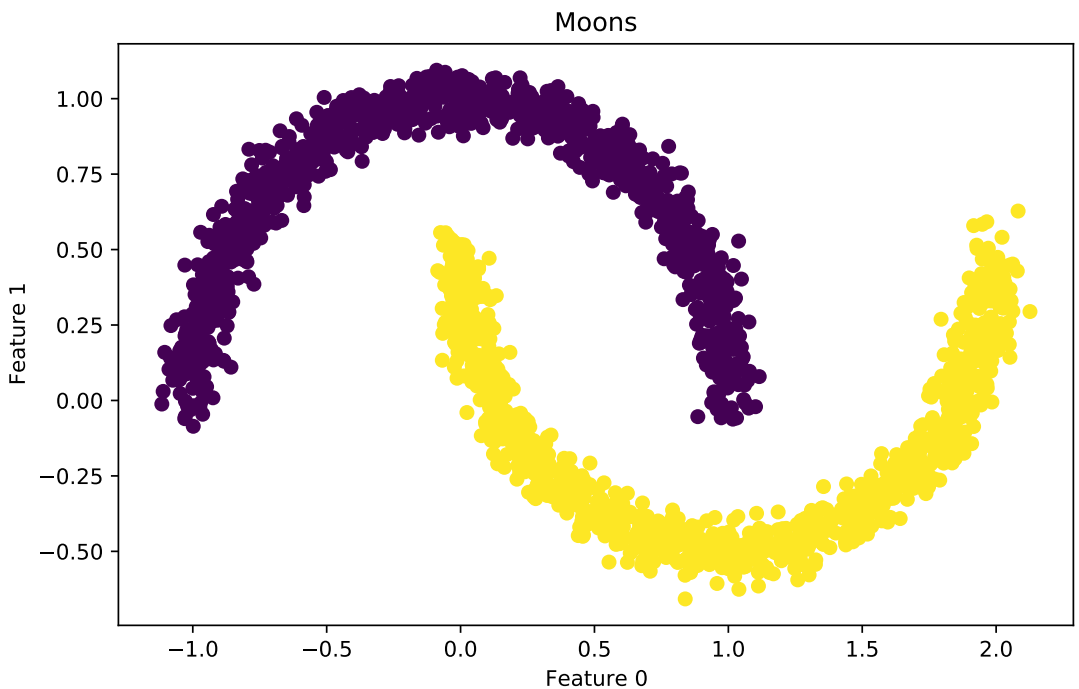
Moons dataset

```
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=2000, noise=0.05)
X.shape  # (2000, 2)
```



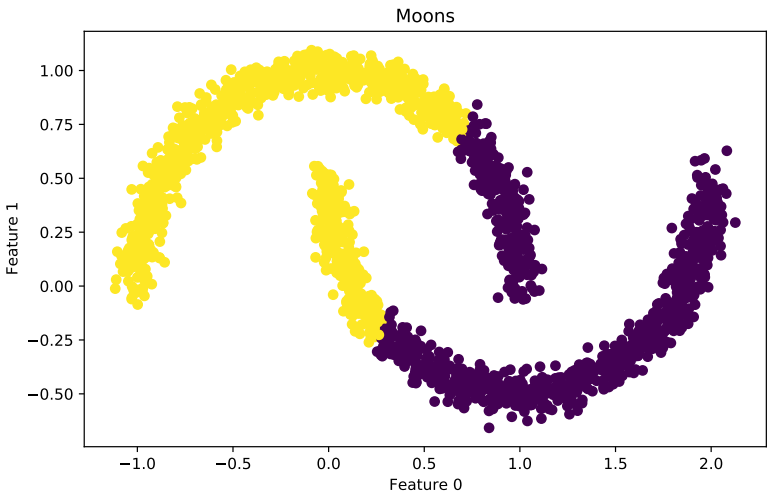
Using label

```
plt.figure(figsize=(8,5))
plt.title("Moons", fontsize=12)
#plt.grid(True)
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.xlabel('Feature 0')
plt.ylabel('Feature 1')
plt.savefig('moons_scatter_colour.pdf', dpi=300)
plt.show()
```



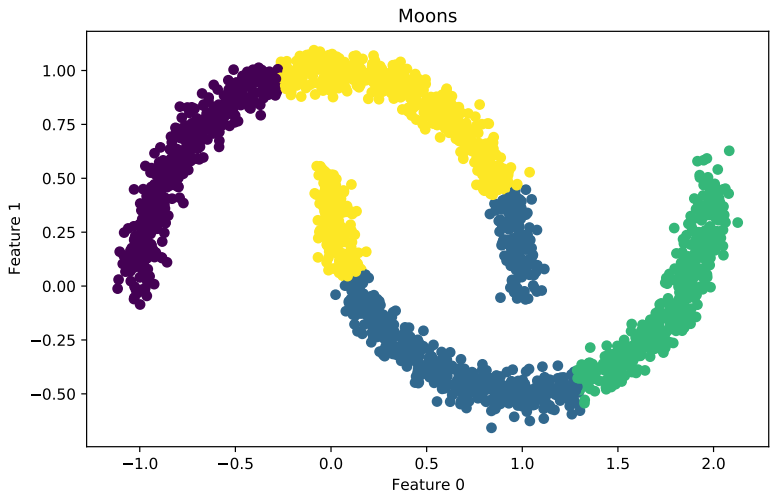
K-Means with varying k

$k = 2$



Completeness score=0.194
Silhouette score=0.489

$k = 4$

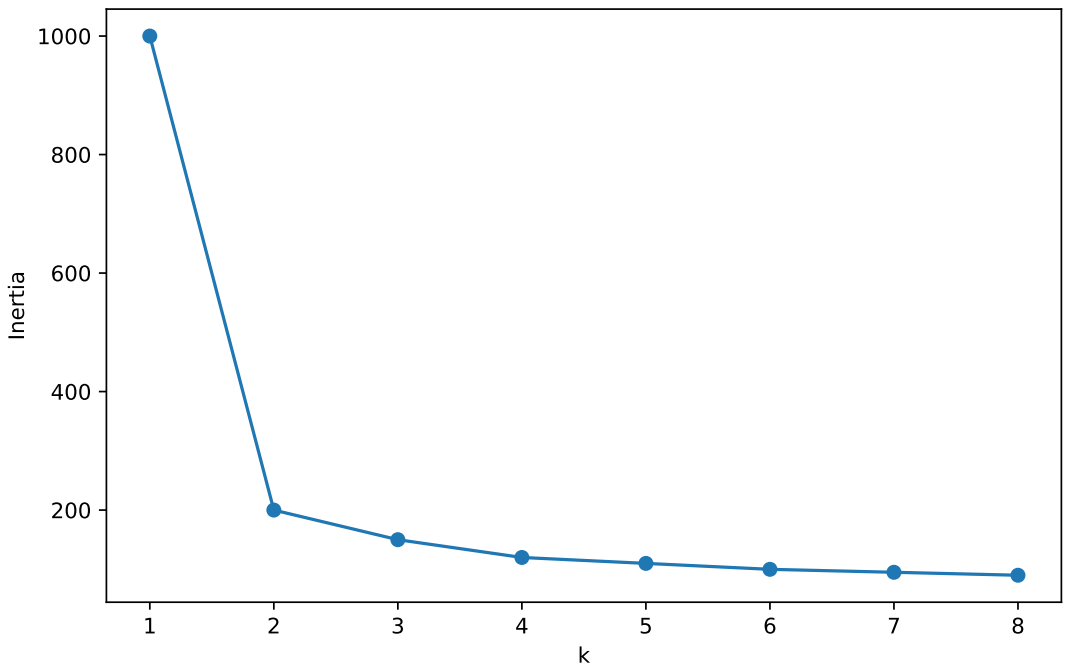


Completeness score=0.273
Silhouette score=0.461

Limitations of K-Means

- Different initial partitions can result in different final clusters
  - Solution: run K-Means multiple times with different initialisations and keep the best solution
    - Inertia – mean squared distance between each instance and its closest centroid
  - Use the n_init parameter (default value is $n\_init = 10$)
- Does not behave well when clusters have different sizes, densities or non-spherical shapes
  - Make sure you scale the features!
  - There are other clustering algorithms we can use
- Need to define k in advance
  - What if we have many dimensions?
  - Use ~~Elbow method~~ silhouette score to find the best value of k

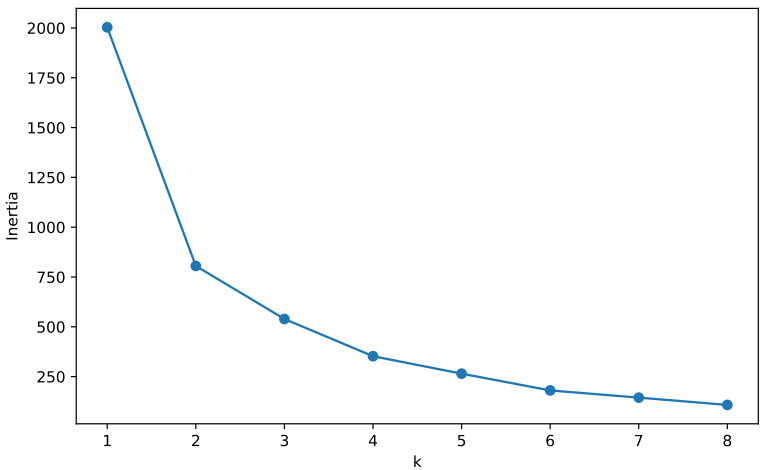
Elbow method

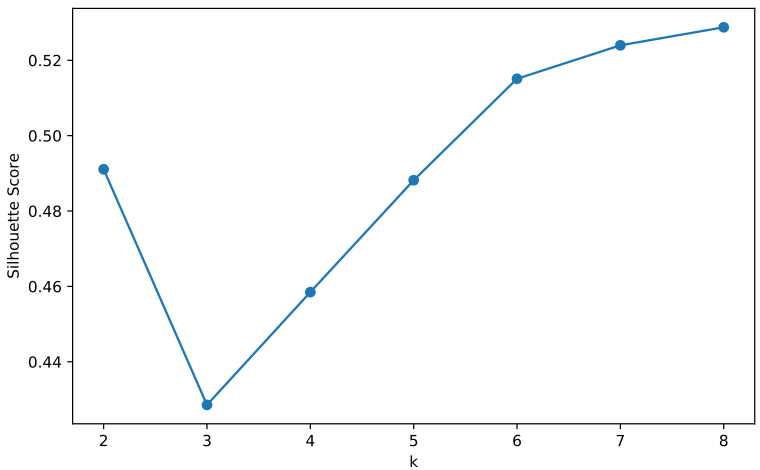


Finding optimal k

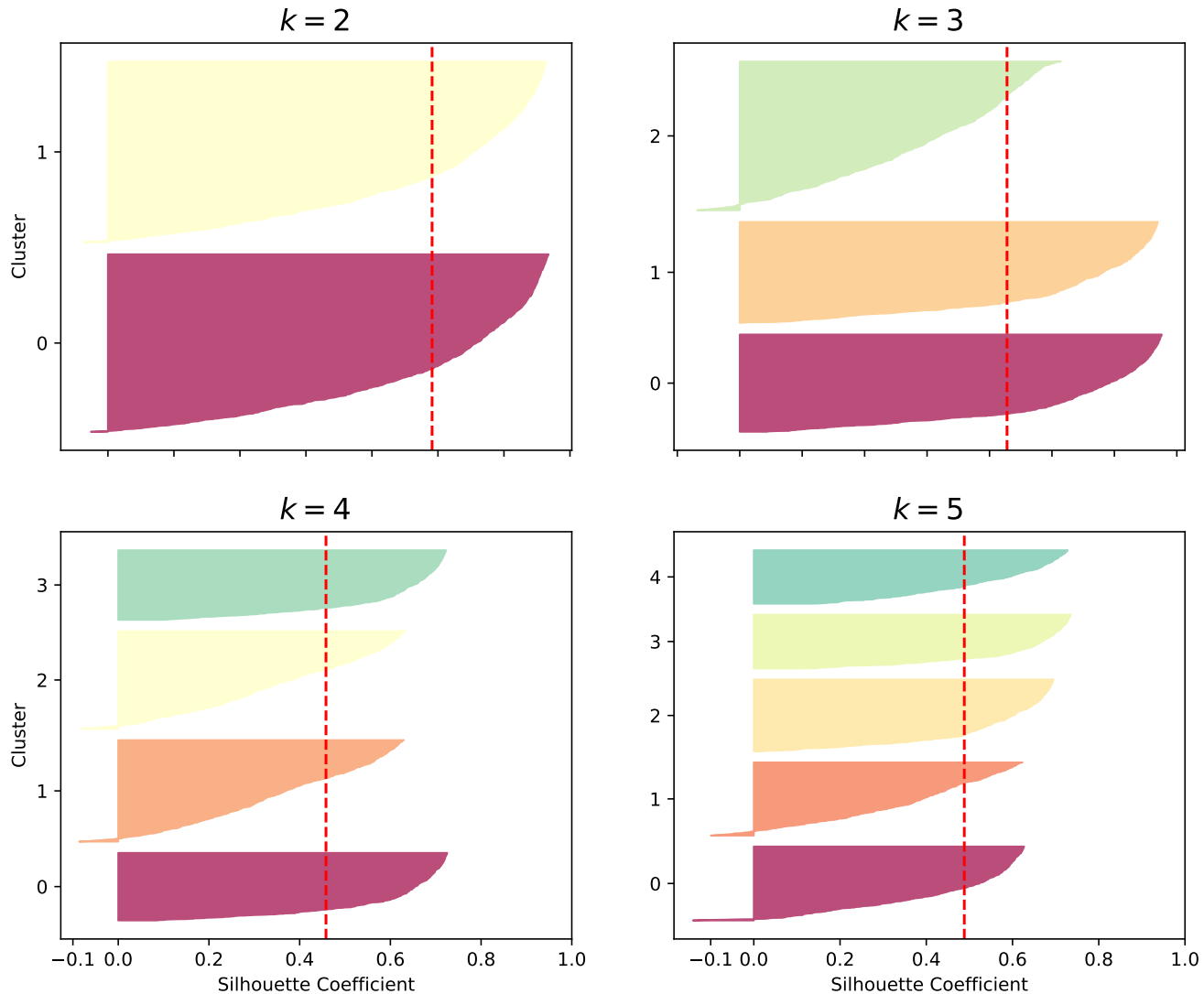
```
inertias = []
for k in range(1, 9):
    kmeans = KMeans(n_clusters=k)
    y_pred = kmeans.fit_predict(X)
    inertias.append(kmeans.inertia_)
# Let's plot inertia vs number of clusters
plt.figure(figsize=(8,5))
plt.plot(range(1, 9), inertias, 'o-')
plt.xlabel('k')
plt.ylabel('Inertia')
plt.show()
```

```
sil = []
for k in range(2, 9):
    kmeans = KMeans(n_clusters=k)
    y_pred = kmeans.fit_predict(X)
    sil.append(silhouette_score(X, y_pred))
# Let's plot SIL vs number of clusters
plt.figure(figsize=(8,5))
plt.plot(range(2, 9), sil, 'o-')
plt.xlabel('k')
plt.ylabel('Silhouette Score')
plt.show()
```





Silhouette diagram



Silhouette diagram (II)

- Red vertical line represents the silhouette score of the whole dataset for each k
- Cluster widths represent the number of data points in each cluster
- To choose k:
  - All clusters within each plot should have SIL bigger than the dataset SIL
  - Cluster sizes should be approximately equal

Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

- DBSCAN defines clusters as continuous regions of high density.
  - Shapes can be arbitrary.
  - Works well when there are high density regions separated by low-density regions.
- Definitions:
  - For each point, count how many other points are located within a distance *eps* from it.
    - This region is the *point's neighborhood*.
  - If there are at least *min_samples* in an instance's neighborhood (itself included), then it's a *core instance*.
  - All points in the neighborhood of a core instance belong to the same cluster – there might be other core instances in the neighborhood.
  - Any instance that is **not** a core instance and is **not** a neighbor of one is an anomaly.

DBSCAN

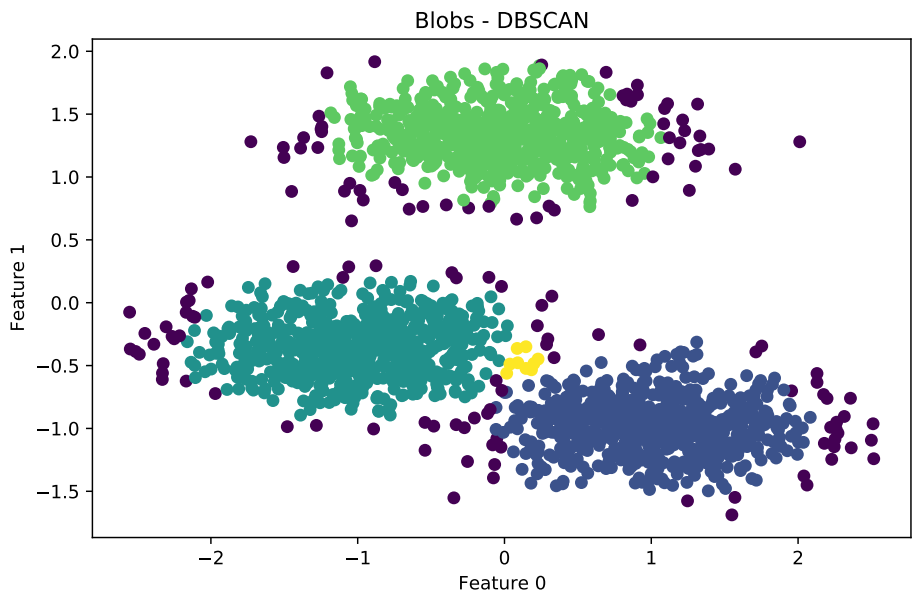
- A cluster is therefore a set of core samples, each close to each other (measured by some distance measure) and a set of non-core samples that are close to a core sample.
- There are two parameters to the algorithm, *min_samples* and *eps*, which define formally what we mean when we say *dense*.
  - Higher *min_samples* or lower *eps* indicate higher density is necessary to form a cluster.

DBSCAN: Advantages and disadvantages

- Advantages:
  - Can discover arbitrarily-shaped clusters
  - Finds clusters completely surrounded by other clusters
- Disadvantages:
  - Datasets with altering densities are tricky
  - Sensitive on two parameters (*min_samples* and *eps*)

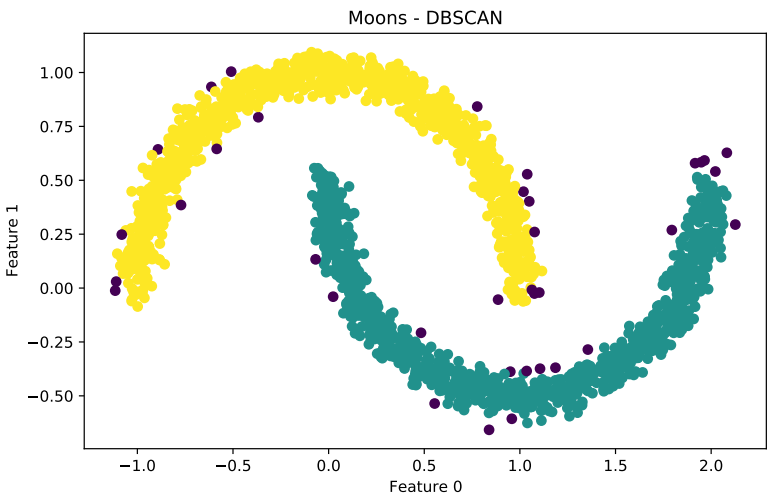
DBSCAN on Blobs

```
from sklearn.cluster import DBSCAN

X_blobs_std = StandardScaler().fit_transform(X_blobs)
dbscan = DBSCAN(eps=0.15, min_samples=10)  # note that k is not a parameter!
dbscan.fit(X_blobs_std)

plt.figure(figsize=(8,5))
plt.title("Blobs - DBSCAN", fontsize=12)
plt.scatter(X_blobs_std[:, 0], X_blobs_std[:, 1], c=dbscan.labels_)
plt.xlabel('Feature 0')
plt.ylabel('Feature 1')
plt.show()
```

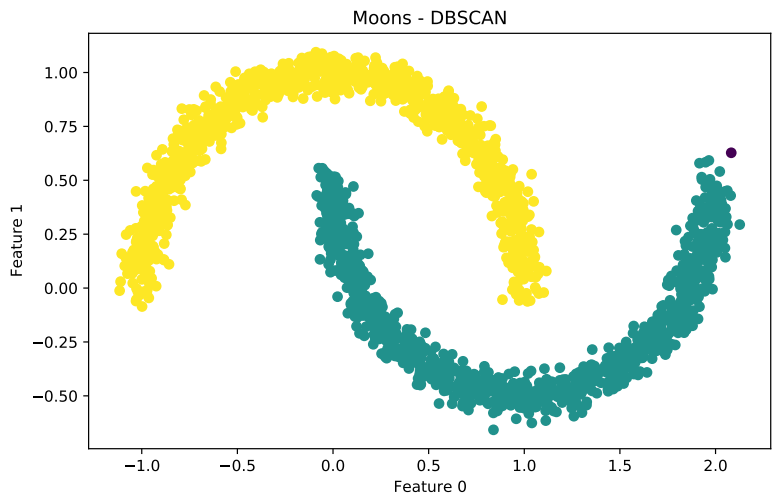


DBSCAN on Moons

```
dbscan = DBSCAN(eps=0.05, min_samples=5)
dbscan.fit(X)
# And plot them
plt.figure(figsize=(8,5))
plt.title("Moons - DBSCAN", fontsize=12)
#plt.grid(True)
plt.scatter(X[:, 0], X[:, 1], c=dbscan.labels_)
plt.xlabel('Feature 0')
plt.ylabel('Feature 1')
plt.show()
```




```
dbscan = DBSCAN(eps=0.1, min_samples=5)
dbscan.fit(X)
# And plot them
plt.figure(figsize=(8,5))
plt.title("Moons - DBSCAN", fontsize=12)
#plt.grid(True)
plt.scatter(X[:, 0], X[:, 1], c=dbscan.labels_)
plt.xlabel('Feature 0')
plt.ylabel('Feature 1')
plt.show()
```



A final note on clustering

- There's no "correct" clustering
- Clusters don't label themselves
  - You have to do that (if you want/have to)

Why?

- Too many features can make training slow
  - Plus, sparsity: predictions are harder to make; larger extrapolations
  - And consequently, higher risk of overfitting
- Useful for data visualisation (2–3 dimensional plots)
  - Essential to communicate results to non-DS people and decision makers
- But...
  - Might lose on performance
    - or not: original features could have been noisy
    - Try first using all features

Principal component analysis (PCA)

- One of the most popular methods of dimensionality reduction
- Emphasizes variation and brings out strong patterns in a dataset.
  - It's often used to make data easy to explore and visualize
- It is an orthogonal transformation to convert a set of correlated variables into a set of values of linearly uncorrelated variables called principal components, with the goal of finding the best summary of the data using a limited number of PCs.
- Finds the hyperplane that lies closest to the data, and projects the data upon it, preserving as much of the variance as possible.
  - By preserving variance, we lose less information.
  - PCs are in order of amount of variance captured.
  - Each component is a linear combination of the input features.
- Quite often followed up by predictions – i.e., we transform the data and use the new features to train a model.

Running PCA

```
from sklearn.decomposition import PCA, KernelPCA

pca = PCA(n_components=2)
X_p = pca.fit_transform(X)
# We can see the explained variance (w.r.t. original dataset) ratio of each component:
print(pca.explained_variance_ratio_)
```

Using PCA

- How many dimensions to keep?
  - For data visualisation: 2-3 PCs
  - For ML: up to x% of variance

Case study: student performance

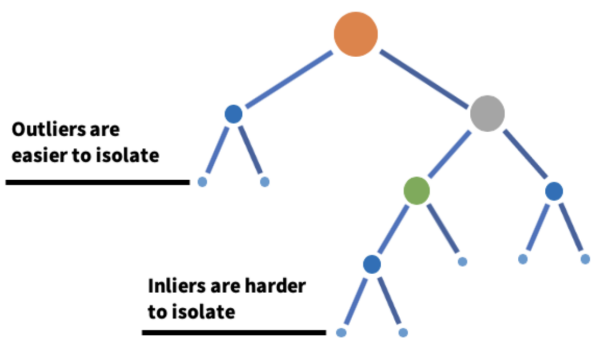
- Let's go to the notebook

Outlier/anomaly detection

- You want to check which observations do not fit in with the rest
- Anomalies are points that are:
  - Few - they are the minority
  - Different - they have values that are very different from those of normal instances
- Isolation forests
  - Since anomalies are "few and different", they are easier to "isolate" compared to normal points
  - Detect anomalies using isolation (how far a data point is to the rest of the data), rather than modelling the normal points
- The algorithm splits the data into two parts randomly until every observation is in its own tree leaf (i.e., isolated)
- Observations with short paths are treated as anomalies
- Very good with high-dimensionality data

Isolation forests

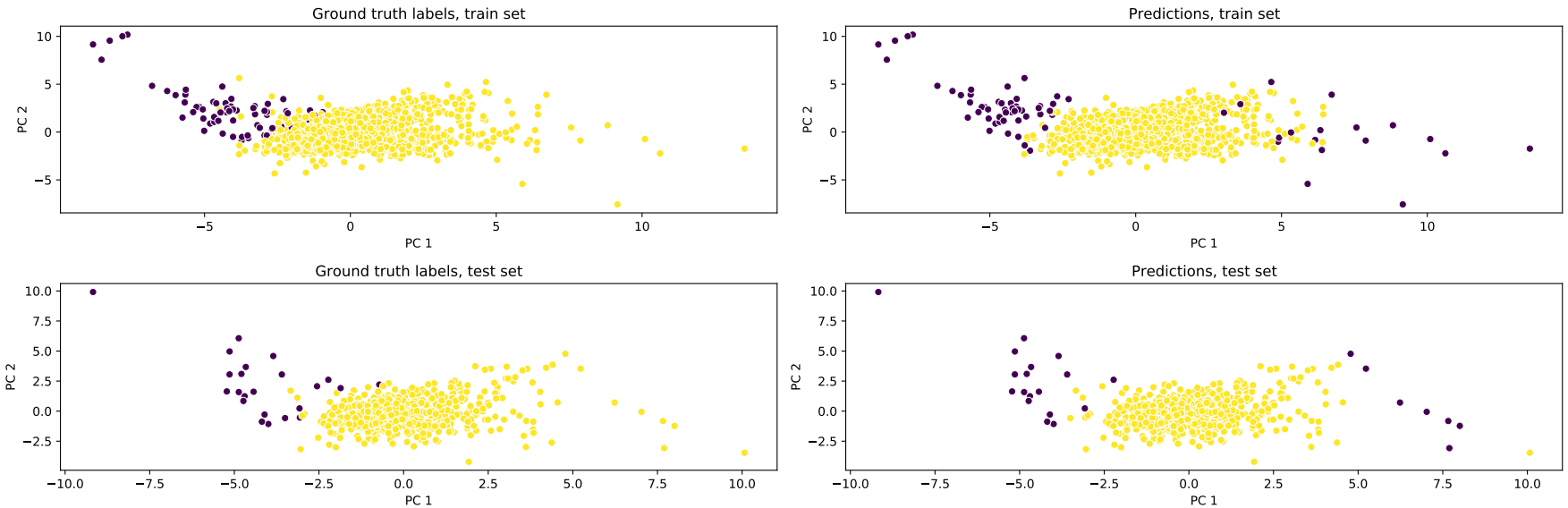
- It builds a random forest using recursive partitioning
- Pick random values and split the data
- Anomalies get isolated faster than normal values by taking fewer steps
- The values that get isolated quicker are anomalies
- It can be used both on numerical and categorical data
- There is no actual "training" or "learning" involved
  - So there is no accuracy test in the conventional ML sense
  - The algorithm uses anomaly scores as a metric, representing the average anomaly score of the input sample



Isolation forests - Code

```
from sklearn.ensemble import IsolationForest
from sklearn.model_selection import train_test_split
from scipy.io import loadmat

# Data is from http://odds.cs.stonybrook.edu/thyroid-disease-dataset/
data = loadmat('./thyroid.mat')
X, y = data['X'], data['y']
# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
                                                    random_state=10, # always get the same splits
                                                    shuffle=True, stratify=y)
# Train an isolation forest with our training set
clf = IsolationForest(max_samples=100, contamination=0.025, random_state=10)
y_pred_train = clf.fit(X_train).predict(X_train)  # Note that we're not passing the labels for training!
y_pred_test = clf.predict(X_test)  # predict on test set
```

Results



Average anomaly scores on train set: standard=-0.421; anomaly=-0.634

Average anomaly scores on test set: standard=-0.420; anomaly=-0.643

Conclusion

- We have seen a set of methods for understanding the data without an outcome signal to guide us
- There's no "correct" clustering, and clusters don't label themselves – you have to do that (if you want to)
- Some methods useful even if you have a label, e.g., do PCA or K-Means on the data and then feed them into a classifier
- Next steps
  - Demos of assignment 1 in this week's labs
    - No demo means 0
  - Make sure you do the quiz to get marked for this unit